MLOps ARTEFACT

Shiv Dutt Singh(L00171329)  
*M.Sc. Big data*Department of Computing  
Atlantic Technological UniversityLetterkenny, Ireland  
l00171329@atu.ie

# Introduction to MLOps

The term "MLOps," which stands for "Machine Learning Operations," refers to the techniques, steps, and equipment used to successfully execute, control, and oversee machine learning models in practical contexts. It blends the core ideas of DevOps (Development Operations) with the issues and difficulties that come up when working with machine learning.

MLOps' main goal is to address the unique requirements and difficulties associated with operationalizing machine learning models. The entire machine learning life cycle, including activities like data preparation, model training, model deployment, monitoring, and maintenance, is its primary area of attention.

MLOps' fundamental objective is to make sure that machine learning models are deployed in a consistent, scalable, and reliable manner while also preserving their performance, accuracy, and robustness over time. Collaboration, reproducibility, scalability, and the seamless integration of machine learning workflows into current software development and operational processes are all given a lot of weight in this approach.

# Research to date on the topic of mlops

Due to the increasing awareness of the importance of efficiently managing and scaling machine learning deployments within enterprises, research on MLOps has taken center stage in recent years. Key areas of study and developing patterns in the world of MLOps include:

1. Model repeatability and Versioning: It is essential for MLOps to ensure the repeatability of machine learning experiments and to maintain version control for models and related data. Most of the research has gone into creating methods and instruments for precisely capturing and managing the many dependencies and configurations required for recreating models. Versioning the runtime environment, hyperparameters, data, preprocessing procedures, and code are all required in this.
2. Automated hyperparameter optimization: The performance of machine learning models is greatly influenced by hyperparameter tuning. To effectively explore the hyperparameter space, research has investigated automated methods like Bayesian optimization and evolutionary algorithms. Model training and deployment can be made more successful and efficient by including these strategies in the MLOps workflow.
3. Continuous Integration and Deployment: These approaches were developed from software engineering to solve the problems that machine learning presents. Automation of the CI/CD process's testing and assessment of machine learning models is the subject of current research. This contains methods for detecting model drift, evaluating the effects of code changes on model performance, and automatically validating models against performance indicators.
4. Model Interpretability and Model Interpretability: these are becoming increasingly important as machine learning models are used more frequently in important fields. To increase knowledge and confidence in deployed models, researchers are proposing techniques to clarify the decision-making process of complicated models. To improve interpretability in MLOps, methods including model-agnostic explanations, rule extraction, and counterfactual explanations are being investigated.

# Description of the ml pipeline implemented

A picture containing text, screenshot, diagram, circle

Description automatically generated

Figure 1: Previous architecture

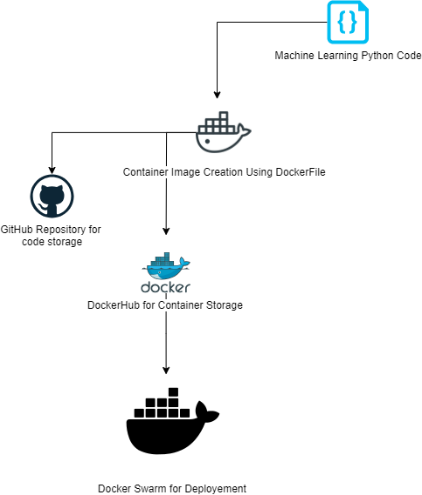


Figure 2: New Architecture

The given project consists of the following with its functionalities.

1. The machine learning model was constructed utilizing the Python programming language. The model was developed through the creation of multiple files using VSCode.
2. The file named "HousePriceModel.py" comprises a machine-learning model that is founded on the principles of linear regression. The data utilized in this study is sourced from the "HousingPriceData.csv" file. The dataset is partitioned into distinct training and testing subsets to assess the model's performance on novel data. The model that has undergone training is serialized and stored as a binary file named "HousePriceModel.pkl," which retains all the essential details required for reproducing the model. This document fulfills two objectives:
   1. The concept of maintaining a model's state or output for future use is commonly referred to as model persistence. The act of preserving the trained model enables its subsequent utilization for generating predictions without necessitating the need for retraining. This feature holds practical value for various applications or for the purpose of model reusability.
   2. The process of converting a model into a format that can be stored or transmitted is known as model serialization. Serialization is a process that converts the model into a binary format, which facilitates its storage and transmission over a network. The model's learned parameters, hyperparameters, and structure are comprehensively captured, thereby guaranteeing its accurate reproduction upon loading.
3. The file named "FlaskApp.py" generates a web-based application programming interface (API) that enables the prediction of car prices by considering the mileage. The Flask web application incorporates a trained model for predicting car prices. The code incorporates essential libraries, including NumPy for numerical computations, request for handling HTTP requests, load from joblib for loading the trained model, and Flask for constructing the web application.

2.1 The instantiation of the Flask class using "Flask(name)" results in the initialization of the Flask application.

2.2 Flask server that exposes two routes: the root route ("/") that displays "Welcome All" and the "/predict" route that accepts query parameters for the house features and returns the predicted house price.

2.3 HouseModelPredication.py prompts the user to enter various features of a house, such as area, number of rooms, bedrooms, bathrooms, stories, and parking spaces.

2.4 Output prints the estimated house price using string formatting to display the value of the price with two decimal places.

2.5 The response message contains the projected value.

1. The file named "HousePriceModel.py" is a program that operates through the command-line interface, and being used to generate file

"HousePriceModel.pkl”.

1. The "Deployment.yaml" file is a configuration file in YAML format for utilizing Kubernetes to deploy a Flask application. The intended state of the "flask-app-deployment" object is specified.
   * The number of application instances is a component of the intended state and is in this case set to 1.
   * The deployment's method for choosing which Pods to govern is defined by the "chooser". It matches the selector in this case that has the label "app: flask-app."
   * "Metadata" includes labels that have been added to each Pod made using the template, with the label "app: flask-app."
   * The "spec" section details the Pod's specifications.
   * The containers to be utilized inside the Pod are described in the "containers" section.
   * The container is known by the moniker "flask-app-container."
   * The Flask application's container image is identified as "docker.io/l00171329/l00171329/mlopsassignment:latest."
   * The port (5000) on which the container listens for incoming traffic is defined in the "ports" section.
2. A configuration for a Kubernetes Service can be found in the "Service.yaml" YAML file that is provided. The "flask-app-service" Service object, meant to expose a Flask application operating in a Kubernetes cluster, is described in the configuration file.

The document is divided into several sections, each with a distinct function:

* apiVersion: Indicates the Kubernetes API version that was utilized in the file. It is currently set to "v1," corresponding to the core/v1 API version.
* kind: Describes the Kubernetes resource type that was declared in the file. It declares a Service in this case.
* Metadata: Consists of information regarding the Service object, such as its name.
* spec: Specifies the ideal condition for the Service object.
* Selector: Defines the criteria for choosing which Pods to direct incoming traffic to. The Service will direct traffic to Pods marked with the label "app: flask-app" in this example because the selector is set to "app: flask-app."
* ports: Defines the ports on which the Service should listen for incoming traffic and direct it to. The protocol used (TCP in this case) is defined by the "protocol" parameter. The listening port, which is set to port 80, is specified by the "port" field.
* targetPort: This specifies the port to which incoming traffic should be forwarded by the Service to reach the chosen Pods. The Flask application within the Pods in this example listens on port 5000.
* type: Identifies the service's type. It is set to "LoadBalancer" in this instance, denoting that the Service should be reachable from the outside world via the load balancer offered by a cloud provider.
* GitHub Repository: To save my code, I've used the GitHub platform. This gives me the ability to tweak my code and utilize it as effectively as possible. It also gives me a good perspective of the modifications that were performed.
* Docker Hub: I created an image of a container and posted it there. Mini Kubernetes will later retrieve this image and use it.

1. I have used docker-compose.yml file for creating deployement using Docker Swarm.

version: '3'

services:

  myapp:

    image: l00171222/mlopsassignment

    ports:

      - "5000:5

* Deployement on Docker Swarm is fairly Easy comparatively than Kubernetes.
* I have use the above file just to create one node which is manager.
* Following command, I have used to deploy

docker stack deploy -c docker-service.yml myapp

1. I have also tried using google VM machine using the same steps I have followed above , but exposing http port for firewall didn’t actually worked for me and I couldn’t able to access public api.

**Discussion of techniques**

I have become proficient in the following methods thanks to class exercises:[4 ]

* Running docker, python, and machine learning models on a Google VM based on Centos 7.
* Developing a Docker container image and submitting it to the Docker Hub.
* Running the ML model on Remote Desktop and evaluating the results.
* Airflow deployment is used.

In this assignment, I used the following resources:

* Utilized an Ubuntu-based Docker Image.
* Used VS Code to create and train an ML model.
* The creation of container images using Docker.
* Use Github to centrally store and make code accessible from wherever.
* Deployment using Mini Kube.

**Analysis of the technologies**

The new tools turned out to be simpler and more effective than the ones I had previously used for ML model deployment. The older technology has various drawbacks that I discovered and have discussed below:

1. I could efficiently manage my code and run it anywhere by utilizing Visual Studio Code. For accessibility, I made sure to submit it to a GitHub repository.
2. My code is now safe thanks to GitHub, which also makes it simple to share it with others as well as just myself.
3. GitHub's authentication procedure is comparatively more complicated for Linux-based systems because, in the Linux terminal, we have to use SSH base authentication, since HTTP prompts are not since Linux use CLI interface (GCP) .
4. I had to use a different technology because the remote desktop connection was having problems. We developed it on local Windows machine, which works with HTTPS and enables us to execute the model locally.
5. There wasn't much of a difference in experience between Airflow and mini-Kubernetes despite the size of the ML model. As seamless as Airflow, mini-Kubernetes were adopted.
6. The minikube dashboard of the future is simple to comprehend and offers a better understanding of deployment. The files deployment.yaml and service.yaml, which provide in-depth insights regarding miniKube deployment, are included.
7. Despite weighing the pros and downsides of MiniKube vs Kubernetes or Airflow, and considering the complexity at hand, we chose MiniKube.
8. MiniKube and Docker Hub communicated with one another without any problems.
9. Overall, the implementation was difficult and called for unconventional thinking. It was very much like a real-life situation.
10. It was captivating and intriguing.

Recommendations and Conclusion

Machine Learning Operations, or MLOps as it is sometimes referred to, concentrates on integrating machine learning models into the software development process. It involves automating processes including model deployment, training, and monitoring.

Python is a popular programming language for data analysis and machine learning. To develop in this area, Visual Studio coding (VS Code) is a well-liked coding editor that offers outstanding Python support.

With the help of the containerization platform Docker, you can package your application and all its dependencies into a standardized unit called a container. The mobility and reproducibility that Docker containers provide make them ideal for deploying machine learning models.

Text files called Docker files contain instructions for creating Docker images. The basic image, dependencies, environment variables, and instructions required to configure your application inside the container are all listed in this document.

A cloud-based registry service called DockerHub allows you to store and share Docker images. It makes it simple to share your application or model with others because it acts as a central hub for pushing and extracting images.

GitHub is a platform for managing and storing your code repositories and is a version control system. Code collaboration, pull requests, and issue tracking are just a few of the capabilities it provides, making it useful for team-based development projects.

With the help of Minikube, you may locally run a single-node Kubernetes cluster for testing and development. Scaling, load balancing, and autonomous container management are some of the functions offered by the orchestration platform known as Kubernetes, which is used to manage containerized applications.  
  
I have also used Google VM and docker Swarm for the deployement in my new assignment, as as said Docker Swarm is fairly easy to use than MiniKube.

Choosing between Ubuntu and CentOS as the foundation image for your Docker container depends on several variables, including your experience with the distribution, compatibility with the dependencies of your application, and the particular needs of your project or company. Both CentOS and Ubuntu have sizable user bases and are frequently used. While CentOS is frequently preferred due to its reliability and long-term maintenance, Ubuntu is well renowned for its user-friendliness and huge package repositories.

In the end, your team and project's unique demands, requirements, and preferences should be taken into consideration while choosing the tools and operating system for your MLOps project. It's crucial to consider elements like compatibility with your application and its dependencies, community support, integration potential, and ease of usage.

##### References

1. Shan, C., Wahba, A., Wang, Li-C. and Sumikawa, N. (2019). *Deploying A Machine Learning Solution As A Surrogate*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ITC44170.2019.9000109.
2. John, M.M., Olsson, H.H. and Bosch, J. (2021). *Towards MLOps: A Framework and Maturity Model*. [online] IEEE Xplore. doi:https://doi.org/10.1109/SEAA53835.2021.00050.
3. Symeonidis, G., Nerantzis, E., Kazakis, A. and Papakostas, G.A. (2022). MLOps - Definitions, Tools and Challenges. *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. doi:https://doi.org/10.1109/ccwc54503.2022.9720902.
4. Makinen, S., Skogstrom, H., Laaksonen, E. and Mikkonen, T. (2021). Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help? *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*. doi:https://doi.org/10.1109/wain52551.2021.00024.
5. John, M.M., Olsson, H.H. and Bosch, J. (2021). *Towards MLOps: A Framework and Maturity Model*. [online] IEEE Xplore. doi:https://doi.org/10.1109/SEAA53835.2021.00050.

‌

SCREENSHOT

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

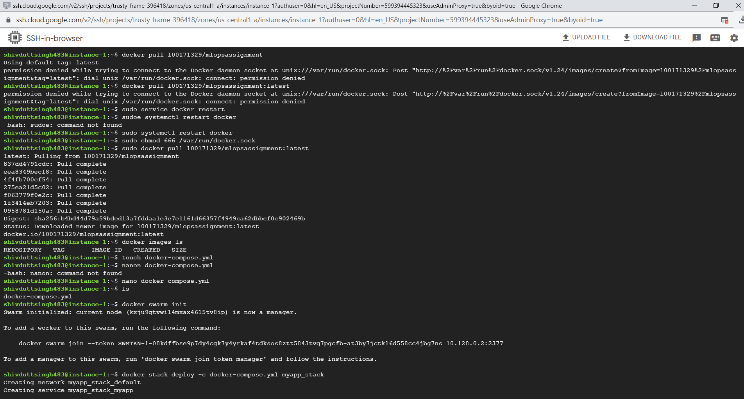
Description automatically generated with medium confidence

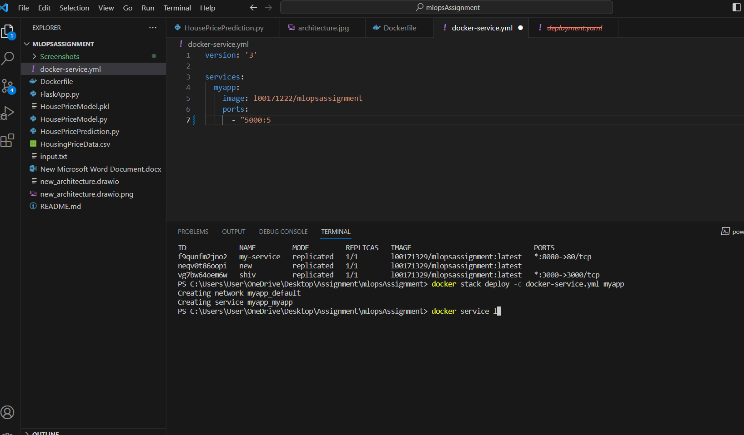
A screenshot of a computer

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated with medium confidence